

3D Printing with Linux and Xenomai

Kendall Auel, Senior Software Engineer, *3D Systems Corp.*


Personal Background

- 36 years developing commercial software
 - 8 years CAD/CAM applications
 - Mechanical CAD/CAM
 - Integrated circuit CAD
 - 28 years embedded software development
 - 3D graphics systems
 - Video processing
 - Printing (2D & 3D)
 - 3+ years at 3D Systems
 - Currently leading the Xenomai technology assessment

Agenda

- Overview of 3D Printing and Multi-Jet Printing Technology
 - Motivation for Studying a Linux / Xenomai Software Solution
- Overview of Xenomai For Real-Time Control
- Typical Use Case – PCI Device Driver
- Practical Experience: Issues and Solutions
- Questions?

3D Printing Overview

- My employer:  3D SYSTEMS®
- Co-founded in 1986 by the inventor of SLA, Chuck Hull
- Added SLS printing in 1989, MJP printing in 1996

SLA	- Stereo Lithography
SLS	- Selective Laser Sintering
MJP	- Multi Jet Printing
CJP	- Color Jet Printing
DLP	- Digital Light Processing
DMP	- Direct Metal Printing
FDM	- Fused Deposition Modeling



MJP – Multi Jet Printing

- Uses inkjet print heads with many hundreds of individual jets
- Object is built layer-by-layer by geometrically slicing a 3D model
- Part material can be plastic (UV-cured) or hard wax
- Support material is soft wax, melted during post-processing



Note: removed video here due to size restrictions.

https://youtu.be/apm5Gn2s_-M

Real-time Subsystems

- Motion Control – X, Y, Z axes, head clean, waste dump
- Thermal Control – Bottles, umbilical lines, print head, fans
- Material Delivery – Level sense, pump and solenoid control
- Print Head Driver – Data load, calibration, jet fire control
- Data Path – Layer data conversion, buffer management

(...While running a compute bound, memory intensive slicer application)

Software Technology Assessment

- Can a single controller provide hard real-time device control, while running a compute-bound slicing application?

We selected Linux with Xenomai, running on a standard x86 micro ATX motherboard.



The Secret Weapon

- Philippe Gerum
 - Consultant, author of Xenomai
- Provided an initial port of our embedded control code to Linux.
- Created an Autotools-based build environment.
- Delivered RTDM drivers and userspace API functions based on the existing implementation.

This was a nice way to start, but it's not difficult to get started on your own...

Getting Started With Xenomai

- Mercury vs. Cobalt (single kernel, or co-kernel configuration)
 - No hard real-time constraints? Choose “Mercury”
 - Any kernel will do
 - Compatible with PREEMPT_RT
 - Hard real-time? Choose “Cobalt”
 - I-Pipe kernel patch
 - Select Linux distribution to be compatible with the required kernel version

Note: Mercury is new for Xenomai-3, Xenomai-2 was strictly co-kernel.

1. Get the kernel source –

<http://www.kernel.org/pub/linux/kernel/v4.x/...>

- Here you can find the specific version for the I-Pipe patch.

or...







```
sudo apt-get source linux-image-$(uname -r)
```

- This gets the kernel version for the distribution you are running.

The first method guarantees your kernel will build and run, while the second method is more compatible with the distribution you're running.
(I've only used the first method, and it's been working without problems.)

<http://xenomai.org/downloads/ipipe/v4.x/x86/>

Index of /downloads/ipipe/v4.x/x86

	Name	Last modified	Size
	Parent Directory		-
	ipipe-core-4.1.18-x86-9.patch	25-May-2017 11:47	451K
	ipipe-core-4.4.43-x86-8.patch	14-Jun-2017 11:47	486K
	ipipe-core-4.4.71-x86-10.patch	03-Oct-2017 12:35	446K
	ipipe-core-4.9.24-x86-2.patch	12-Jun-2017 11:06	491K
	ipipe-core-4.9.38-x86-4.patch	03-Oct-2017 12:41	452K
	ipipe-core-4.9.51-x86-4.patch	03-Oct-2017 15:46	452K
	older/	03-Oct-2017 12:41	-

Ubuntu 14.04.5:
- Linux 4.4.0

Ubuntu 16.04.3
- Linux 4.10.0

2. Get the xenomai source –

`git clone git://git.xenomai.org/xenomai-3`

or...

Download:

<http://xenomai.org/downloads/xenomai/stable/xenomai-3.0.6.tar.bz2>

Cloning the repository is best. You'll be able to get updates very easily.

3. Build xenomai using autotools

- bootstrap, configure, make, make install

4. Copy the I-Pipe patch script into the kernel source directory

- `wget http://www.xenomai.org/downloads/ipipe/v4.x/x86/...`

or...

- `curl http://xenomai.org/downloads/ipipe/v4.x/x86/... -o ipipe-core-....`

Note: I've found that using curl is more reliable than wget.

5. Run the xenomai prepare-kernel.sh script to patch the kernel
6. Configure the kernel, typically using menuconfig
 - This can be the most confusing and difficult part of the build process, especially if you haven't done much kernel development.

```
.config - Linux/x86 4.1.18 Kernel Configuration

Linux/x86 4.1.18 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus
----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M>
modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search.
Legend: [*] built-in [ ] excluded <M> module < > module capable

[*] 64-bit kernel (NEW)
  General setup --->
    [*] Enable loadable module support --->
    -- Enable the block layer --->
    [*] Xenomai/cobalt (NEW) --->
      *** WARNING! Page migration (CONFIG_MIGRATION) may increase ***
      *** latency. ***
      *** WARNING! At least one of APM, CPU frequency scaling, ACPI 'processo
      *** or CPU idle features is enabled. Any of these options may ***
      *** cause troubles with Xenomai. You should disable them. ***
      Processor type and features --->

+(<+>
<Select> < Exit > < Help > < Save > < Load >
```

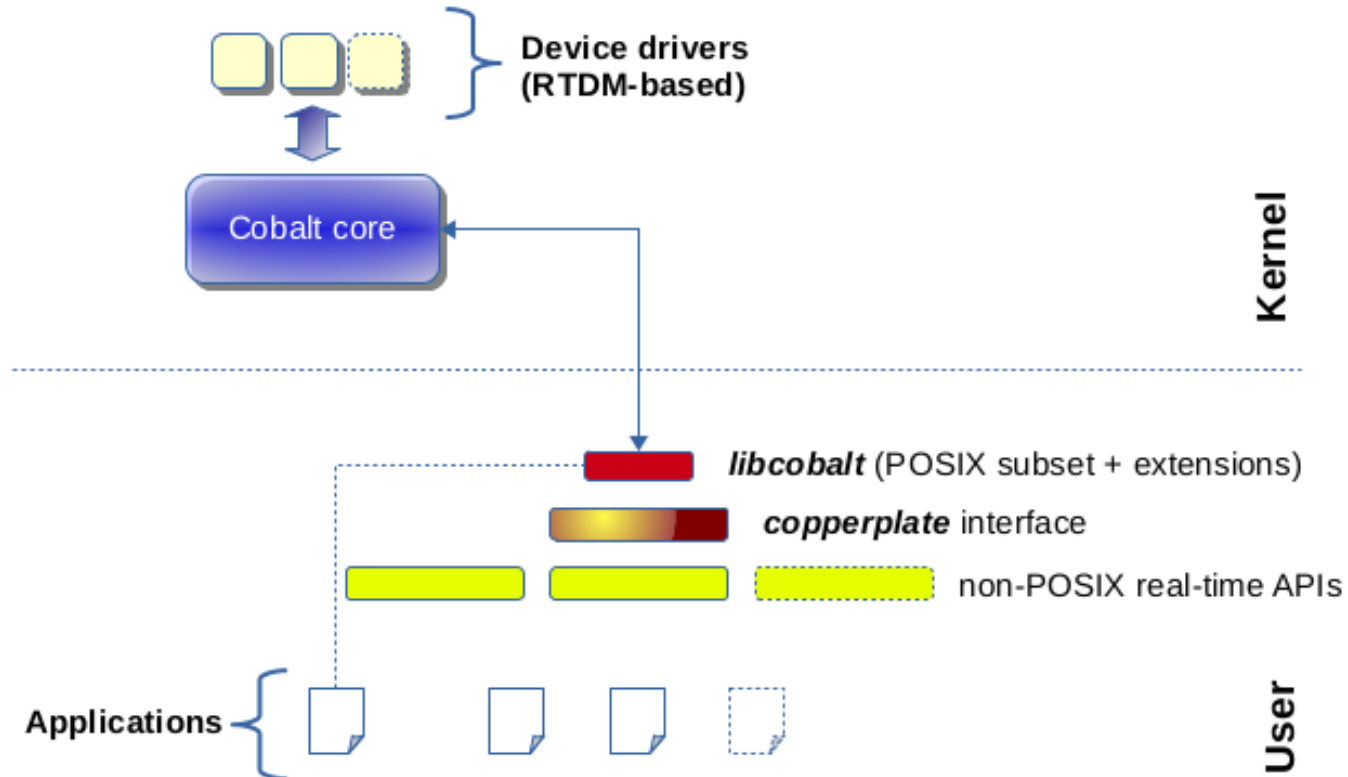
```
Enable the block layer --->
[*] Xenomai/cobalt (NEW) --->
  *** WARNING! Page migration (CONFIG_MIGRATION) may increase ***
  *** latency. ***
  *** WARNING! At least one of APM, CPU frequency scaling, ACPI 'processo
  *** or CPU idle features is enabled. Any of these options may ***
  *** cause troubles with Xenomai. You should disable them. ***
  Processor type and features --->
```

- 7. Make and install the kernel as usual
 - Nice to have a “Local version” string appended to kernel release
- 8. Verify Xenomai is functioning
 - /usr/xenomai/bin - contains a variety of test functions
 - /usr/xenomai/bin/latency - basic test to verify xenomai is working

```
0"000.004| WARNING: [main] Xenomai compiled with partial debug enabled,  
                                high latencies expected [--enable-debug=partial]  
== Sampling period: 100 us  
== Test mode: periodic user-mode task  
== All results in microseconds  
warming up...  
RTT| 00:00:01 (periodic user-mode task, 100 us period, priority 99)  
RTH|----lat min|----lat avg|----lat max|-overrun|---msw|---lat best|--lat worst  
RTD|    0.059|    0.250|    47.151|    0|    0|    0.059|    47.151  
RTD|    0.105|    0.243|    45.117|    0|    0|    0.059|    47.151  
RTD|    0.102|    0.259|    56.229|    0|    0|    0.059|    56.229  
RTD|    0.026|    0.249|    25.561|    0|    0|    0.026|    56.229  
RTD|    0.100|    0.254|    52.520|    0|    0|    0.026|    56.229  
RTD|   -0.002|    0.251|    51.588|    0|    0|   -0.002|    56.229  
RTD|   -0.063|    0.247|    28.560|    0|    0|   -0.063|    56.229  
RTD|    0.105|    0.252|    53.369|    0|    0|   -0.063|    56.229  
RTD|    0.075|    0.255|    24.659|    0|    0|   -0.063|    56.229  
RTD|    0.053|    0.255|    55.168|    0|    0|   -0.063|    56.229
```

- *Note: All Xenomai applications must be run as superuser.*

Xenomai Co-Kernel Architecture (Cobalt)



Device Drivers and the Real Time Driver Model

- RTDM: unified interface for users and developers of RT drivers
 - Adopted as the Xenomai driver programming interface
 - Also used by the Real Time Application Interface (RTAI)
 - Politecnico de Milano, Italy
- RTDM devices are found under `/dev/rtdm`
- Register IRQ handler functions
- Register devices (open / close / read / write / ioctl / select etc.)

Typical Use Case – PCI Device

- Get ownership of a PCI device (by vendor_id / device_id pair)
 - `pci_enable_device(pci_dev *dev)`
 - `pci_request_regions(pci_dev *dev, const char *res_name)`
 - `pci_resource_start(pci_dev *dev, int bar)` // base address register
 - `pci_ioremap_bar(pci_dev *dev, int bar)`
- Set up interrupt callback
 - `rtdm_irq_request(.... dev->irq, rtdm_irq_handler_t handler)`

Typical Use Case – Continued

- Register the device as an RTDM driver
 - **fpga_class** = `class_create(THIS_MODULE, “fpga”)`
 - `rtdm_drv_set_sysclass(&fpga_driver, fpga_class)`
 - `rtdm_dev_register(&fpga_device)`
- The RTDM device structure references the driver structure:

```
struct rtdm_device fpga_device = {  
    .driver = &fpga_driver  
    .label = “fpga/control”  
}
```

```
static struct rtdm_driver control_driver = {
    .profile_info = RTDM_PROFILE_INFO(fpga_control,
                                      RTDM_CLASS_FPGA,
                                      RTDM_SUBCLASS_FPGA_CONTROL,
                                      0),
    .device_flags = RTDM_NAMED_DEVICE,
    .device_count = 1,
    .context_size = sizeof(struct fpga_context),
    .ops = {
        .open      = control_open,
        .close     = control_close,
        .ioctl_rt  = control_ioctl_rt,
        .ioctl_nrt = control_ioctl_nrt,
        .write_rt  = control_write,
        .read_rt   = control_read,
        .mmap      = control_mmap,
    },
};
```

Driver ops include real-time (rt) and non-real-time (nrt) versions of ioctl, write, read, recvmsg, and sendmsg.

RTDM drivers can run in primary mode, or secondary mode.

Note: Beware locking RT resources in secondary mode.

Create a POSIX-Compatible Clock

- Populate xnclock struct, including operations for:
 - read_raw (hardware ticks) and read_monotonic (nanoseconds)
 - set_time (given current time in nanoseconds)
 - ns_to_ticks, ticks_to_ns, and ticks_to_ns_rounded
 - program_local_shot, program_remote_shot (to set interrupt time)
- Call xnclock_tick(struct xnclock *clock) from interrupt handler
- Register with cobalt_clock_register(struct xnclock *clock,)
 - Returns POSIX clockid_t
 - Use clockid_t for timer_create, timer_settime, clock_nanosleep, ...

Write a User Space Application

- Open your device:
 - `fd = open("dev/rtdm/fpga/control", O_RDWR);`
 - Same as: `__RT(open("dev/rtdm/fpga/control", O_RDWR));`
 - Xenomai symbol wrapper script for the POSIX API does this for you
- Read / Write data:
 - `nr = read(fd, buff, blen);` `// or: __RT(read(fd, buff, blen));`
 - `nw = write(fd, buff, blen);` `// or: __RT(write(fd, buff, blen));`
- Start a new real-time thread:
 - `__RT(pthread_create(&tid, &attr, thread_main, thread_params));`

Thread synchronization and communication

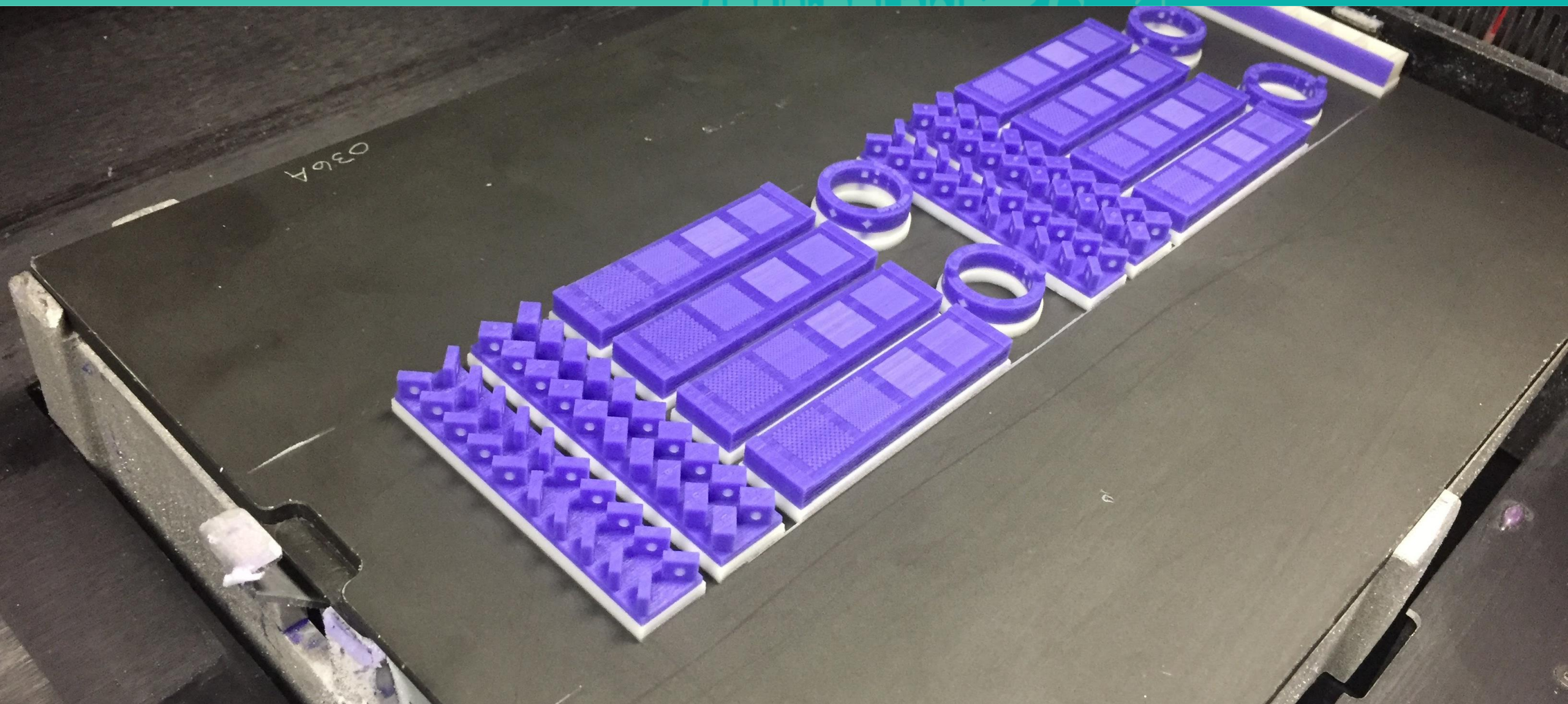
- Mutexes

```
__RT(pthread_mutex_lock( ... ));  
__RT(pthread_mutex_unlock( ... ));
```

- Condition variables

```
__RT(pthread_mutex_lock( ... ));  
while ( ! data_ready ) {  
    __RT(pthread_cond_wait( ... ));  
}  
__RT(pthread_mutex_unlock( ... ));
```


Note: removed video here due to size restrictions.



Practical Experience – Issues and Solutions

- Thread “relaxes”: unexpected switch to secondary mode
- System lockup calling ‘gettimeofday’ in primary mode
- Deadlock in kernel driver periodic function (nklock ordering)
- Timer event arriving before the requested time

Thread Relaxes

- Threads run as 'primary' or 'secondary' mode
- Transition from primary to secondary is called a 'relax'
- Some relaxes are 'spurious', due to unexpected side-effects
- Examples:
 - Auto-grow a collection class object (malloc from Linux heap)
 - Dereference a pointer to memory-mapped file data

Relax Due To File I/O

- Non-volatile parameters are kept in a file
- Reading and writing the file causes primary mode to relax
- Solution:
 - Use in-memory file cache
 - Update file via XDDP socket (Cross Domain Datagram Protocol)

Calling 'gettimeofday' In Primary Mode

- Almost always works fine
- On rare occasions, Linux OS is interrupted while updating
- Caller spin-locks waiting for update to complete
- Linux can't run because primary mode is spinning
- Solution:
 - `__RT(clock_gettime(CLOCK_HOST_REALTIME, &ts));`
 - `nanosecs_abs_t ns = rtdm_clock_read();` *// for kernel drivers*

Kernel Periodic Function – nklock Deadlock

- Assume we have a shared resource – FPGA
 - Protected with RTDM spinlock
- Normal operation:
 1. Lock FPGA resource: `rtdm_lock_get_irq_save(&fpga_lock, ctx);`
 2. Signal for select call: `xnselect_signal(&sel, POLLIN);` // locks nklock
- Periodic timer function runs with nklock locked:
 1. Lock FPGA resource

Deadlock!

Avoid nklock Deadlock

- Never call RTDM or xn... functions with resource locked

How do we guarantee nothing in the call tree locks nklock?

or...

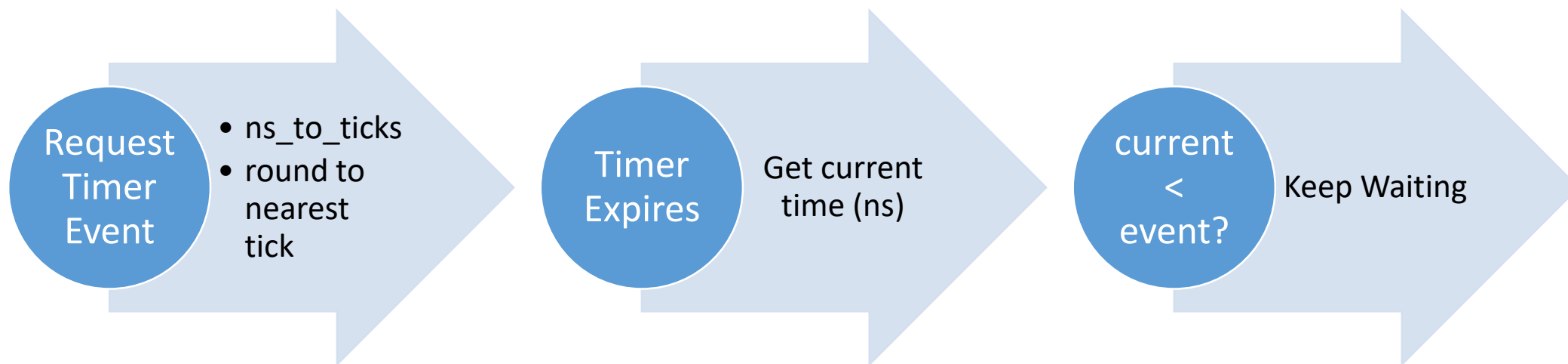
- Never lock FPGA resource within a periodic timer function

How do we safely utilize the FPGA resource on a periodic basis?

or...

- Build timer interrupt into your FPGA (?)

Timer Event Arrives Early



On the original system, minimum latency was greater than a tick, so we never saw [current < event]. The Xenomai latency is small enough to create this issue.

➤ **Solution: add one tick to the current time before comparison.**

Results

- Embedded control code base now runs on Linux / Xenomai.
- We can also build and run for 32-bit, commercial RTOS.
- Slicer runs as standard Linux application
 - Consumes 4 cores, 16 GB DDR.
- Real-time control application gets short and bounded latencies.

Questions?



Embedded Linux
Conference
North America



OpenIoT Summit
North America